A parallel two-level preconditioner for Cosmic Microwave Background map-making

Laura Grigori
INRIA Saclay-Île de France
F-91893 Orsay, France
Email: laura.grigori@inria.fr

Radek Stompor
APC, Univ Paris Diderot, CNRS/IN2P3,
CEA/Irfu, Obs de Paris, Sorbonne Paris Cité, France
Email: radek@apc.univ-paris-diderot.fr

Mikolaj Szydlarski
INRIA Saclay-Île de France
F-91893 Orsay, France
Email: mikolaj.szydlarski@inria.fr

***Abstract*—Generalized least square problems with non-diagonal weights arise frequently in an estimation of two dimensional images from data of cosmological as well as astro- or geo- physical observations. As the observational data sets keep growing at Moore's rate, with their volumes exceeding tens and hundreds billions of samples, the need for fast and efficiently parallelizable iterative solvers is generally recognized.**

In this work we propose a new iterative algorithm for solving generalized least square systems with weights given by a block-diagonal matrix with Toeplitz blocks. Such cases are physically well motivated and correspond to measurement noise being piece-wise stationary – a common occurrence in many actual observations. Our iterative algorithm is based on the conjugate gradient method and includes a parallel two-level preconditioner (2lvl-PCG) constructed from a limited number of sparse vectors estimated from the coefficients of the initial linear system.

Our prototypical application is the map-making problem in the Cosmic Microwave Background data analysis. We show experimentally that our parallel implementation of 2lvl-PCG outperforms by a factor of up to 6 the standard one-level PCG in terms of both the convergence rate and the time to solution on up to $12,228$ cores of NERSC's Cray XE6 (Hopper) system displaying nearly perfect strong and weak scaling behavior in this regime.

I. Context

An estimation of a signal from a sequence of noisy data is one of the keystone problems in modern cosmology as well as many other scientific and practical applications. In cosmology, the problem, referred to commonly as a map-making, has come to the fore with the advent of new highly sensitive detectors which have permitted high density sampling of the sky over large areas. As of today, such observations are performed in a number of spectral frequency bands with variety of detectors and instrumental set-ups. They all however share a similar problem. This is that of efficient sky signal recovery from a series of noisy measurements.

In this work, for definiteness, we focus on the Cosmic Microwave Background (CMB) observations and investigate the map-making problem in this context. CMB research has been a major driving force behind the revolution in our understanding of the Universe, its evolution and composition which has been taking place over the past two decades. It studies minute fluctuations imprinted on otherwise homogeneous background of photons produced in the very early Universe. As such CMB provides at this time one of the most direct and efficient ways available to us to get insights about the early Universe and constraints some of its most fundamental parameters. The smallness of the observed signals, as compared to the instrumental noise level, unavoidably drives volumes of the data, which need to be collected to permit explorations of progressively more subtle and demanding science goals. Indeed, over the past decade the sizes of the CMB data sets have kept growing at the Moore's rate [1] constantly requiring computer-processing power corresponding to the largest supercomputers at that time. The current data sets are composed of many tens of Terabytes of the unprocessed data and Petabyte sets are already anticipated.

CMB experiments typically scan the sky for long periods of time (from weeks to years), with thousands of detectors, registering the measurements at a rate on order of milliseconds, while covering extensive sky areas. A standard CMB data set is therefore made of time series of data composed of as many as $\mathcal{N}_t \sim \mathcal{O}(10^{10} - 10^{11})$ measurements and encodes information about sky signal amplitudes corresponding to as many as $\mathcal{N}_p \sim \mathcal{O}(10^6 - 10^7)$ independent, resolution-size, sky pixels.

The map-making in the CMB context aims at producing sky maps of the CMB fluctuations and is therefore one of the fundamental steps down the data analysis pipeline of any CMB experiment. However, map-making-like operations are significantly more ubiquitous in the CMB analysis and have a direct, crucial impact on multiple other stages of the processing. These range from a component separation procedure to a power spectrum estimation, where often thousands of the map-making operations have to be performed, e.g., in some sampling algorithms, to derive a final result. At the same time the minute amplitudes of the sought-after signals and tremendous sizes of the current and anticipated CMB data sets conspire to make this operation particularly challenging. This difficulty is further enhanced by the fact that the CMB detectors are susceptible to long-term thermal drifts inducing strong correlations between the measured samples, which need to be accounted for if high quality maps are to be produced.

In this paper we consider the CMB map-making as a generalized least square problem and discuss a construction of new preconditioning techniques for an iterative, preconditioned conjugate gradient algorithm we use to solve it. The challenges here are manifold as the successful preconditioners not only have to accelerate algorithm's convergence in terms of the number of iterations required to the solution, but also

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

have to be cheap from the point of view of the computation as well as intra-process communication, not to erase the performance benefit arising from the decreased number of iterations. This work shows that such challenges can be indeed met successfully resulting in important speed-ups of the map-making solvers.

The structure of this paper is as follows. In the next Section we introduce an algebraic framework within which we consider the map-making problem. In section III we present our preconditioner and in section IV describe its parallel implementation. Our numerical experiments are presented in section V and our main results summarized in section VI.

II. Problem definition

An observational datum, d_t, registered by a generic detector at time t can be generally modeled as a sum of two contributions, one due to the sky signal and the other due to the instrumental noise $\breve{n}_t$,

$$d_t = \mathcal{A}_t \star \bar{x} + \breve{n}_t = (\mathcal{B} \star \mathcal{R}_{\gamma_t}) \star \bar{x} + \breve{n}_t. \tag{1}$$

Here, the star denotes a convolution of a kernel, $\mathcal{A}_t$, and a true underlying sky signal, $\bar{x}$. Hereafter, we will refer to the vector d_t as time-domain data. The convolution kernel in general depends on an observation direction, γ_t, at time t and an instrumental response, and can be represented as a convolution of a rotation operator, $\mathcal{R}_{\gamma_t}$, and the instrumental response, $\mathcal{B}$, with the former actively rotating the signal, $\bar{x}$, to coordinates in which the latter is defined. $\mathcal{R}_{\gamma_t}$ in general is a function of three Euler angles. In cases in which the response is axially symmetric with respect to the observation direction the detected signal can be alternately modeled as

$$d_t = \mathcal{P}_t \star \bar{x}_B + \breve{n}_t \tag{2}$$

where for any direction γ, $\bar{x}_B(\gamma) \equiv (\mathcal{B} \star \mathcal{R}_\gamma) \star \bar{x}$ is a response convolved signal, which, given the assumption, is a unique function of the direction and therefore has the same number of degrees of freedom as the original signal, $\bar{x}$. The operator $\mathcal{P}_t \propto \delta_{Dirac}(\gamma_t - \gamma)$ defines the observation direction for each registered datum. In a discretized form in which the signal is to be estimated only in a finite number of pixels, the latter equation then reads, e.g., [2], [3],

$$d = Px + \breve{n}. \tag{3}$$

Here x is a discretized version of $\bar{x}_B$ and the matrix P – that of the operator $\mathcal{P}$. If we denote by $\mathcal{N}_t$ a number of time domain data samples and by $\mathcal{N}_p$ ($\ll \mathcal{N}_t$) a number of pixels, the matrix P is tall and skinny and moreover very sparse. In fact, given our assumptions, it has a single non-zero entry for each row, which is always equal to one. The column number of such a non-zero entry defines a number of a pixel observed at a time as indexed by the entry's row number. In this way the matrix P encodes a sky scanning strategy, i.e., the way in which the sky has been observed during the observation. The matrix P is full column-rank. In more general cases, in which the measured data are linear combination of signals of different physical origins and which are to be separated in the estimation process, the matrix P may possess more non-zeros per row than just one. However, we restrict ourselves here to a former case, in which the standard preconditioner has been found to perform particularly well, setting the bar very high for any attempt to introduce better more efficient preconditionning prescriptions, and therefore a demanding and telling test case for investigations as the one undertaken in this work.

The time-domain vector $\breve{n}$ denotes the instrumental noise. Such noise is typically well described as a Gaussian and piecewise stationary process, with covariance $N = \langle \breve{n}\, \breve{n}^T \rangle$ given as a block-diagonal matrix with Toeplitz blocks, each corresponding to a different stationary time piece. Though an inverse of a Toeplitz matrix is not Toeplitz, it has been shown [4] that with precision sufficient for most of the practical purposes N^{-1} can be approximated as a block-diagonal matrix with Toeplitz blocks, which are constructed directly from the corresponding blocks of the covariance itself. Further approximations are also possible [4], e.g., Toeplitz blocks are frequently assumed to be band-diagonal, and often employed to accelerate the numerical calculations, but are not used in this work. We thus can write,

$$N^{-1} = \begin{pmatrix} T_1 & 0 & \dots & 0 \\ 0 & T_2 & \dots & 0 \\ \vdots & \dots & \ddots & 0 \\ 0 & \dots & \dots & T_K \end{pmatrix}, \tag{4}$$

where T_i denotes a symmetric Toeplitz block with a dimension t_i and therefore $\sum_{i=1}^{K} t_i = \mathcal{N}_t$. The noise covariance (as well as its inverse) is a matrix of size $\mathcal{N}_t \times \mathcal{N}_t$ and clearly prohibitively large to be stored in modern computer memory ($\mathcal{N}_t \gtrsim \mathcal{O}(10^9)$. However if its structure is accounted for, the required memory for its storage is found to be the same as needed to store the data vector, d. This is because each symmetric Toeplitz block is uniquely defined by its first row.

The problem of reconstructing the underlying sky signals amounts therefore to solving efficiently the inverse problem to that defined by the overdetermined linear system defined in equation (3). Moreover, given the sizes of the time domain data, this has to be done in a massively parallel way. From the numerical perspective, e.g., [5], ideally the solution is obtained via an explicit QR decomposition of the system matrix, P. Such an approach applied directly would however ignore the weights given by the noise covariance and typically result in a poor quality estimate. To correct for that the QR decomposition should be rather applied to a "prewhitened" system, i.e.,

$$N^{-1/2}\, d = N^{-1/2}\, Px + N^{-1/2}\ \breve{n}. \tag{5}$$

Though the noise weights in this case are by construction diagonal the system matrix is now given by $N^{-1/2}\, P$ and therefore generally not sparse making the QR decomposition prohibitive due to the sizes of the CMB data sets.

An alternate approach is to look at the problem as a generalized least square problem, e.g., [5], [3], with the solution given by,

$$\widetilde{x} = W\, d. \tag{6}$$

where

$$W = \left[P^T N^{-1} P\right]^{-1} P^T N^{-1}. \quad (7)$$

A direct evaluation of such a solution is however again infeasible as it requires an explicit construction and inversion of a dense, square matrix of a rank $\mathcal{N}_p \times \mathcal{N}_p$,

$$S \equiv P^T N^{-1} P, \quad (8)$$

and both of these operations are readily prohibitive given the sizes of involved data objects.

The problem can be however rephrased as a linear system with a square, positive-definite coefficient matrix S, e.g. [1],

$$S\,\widetilde{x} \;=\; b \quad (9)$$

where

$$b \;\equiv\; P^T N^{-1}\, d. \quad (10)$$

and the looked for solution can be then derived with help of some iterative linear system solver. Iterative solvers generically perform repetitively matrix-vector products of the system coefficient matrix, here S, and a vector. The question therefore arises if that can be accomplished without ever explicitly constructing S. This indeed turns out to be possible by representing S as in equation (8) and performing the involved matrix-vector operations from right to left, e.g., [1]. A similar approach also permits a computation of the vector b via equation (10).

The applicability of this approach is hinged on two facts. First, we require efficient, parallel algorithms to perform a matrix-vector product of the noise covariance matrix, N^{-1} and an arbitrary time domain vector (weighting), a matrix-vector product of the sparse matrix P^T by an arbitrary time domain vector (pointing), and a matrix-product of the sparse matrix, P by any pixel-domain vector (unpointing). Second, we require an efficient, quickly converging iterative solver. To date, the first issue above has been extensively studied and high performance solutions have been developed and implemented, e.g., [1], [6]. By contrast the second issue has been left largely intact with nearly all performed studies relying on the preconditioned conjugate gradient (PCG) solver and invoking the same, intuitive, easy to compute and implement preconditioner [1], given as,

$$D^{-1} = (P^T \,\mathrm{diag}\,(N^{-1})\, P)^{-1}. \quad (11)$$

Here $\mathrm{diag}\,(N^{-1})$ is a diagonal matrix consisting of the diagonal elements of N^{-1}. D^{-1} coincides with the inverse of the coefficient matrix of our system, S, whenever the time domain noise is white and therefore can be expected to provide a satisfactory approximation to S^{-1} also in more general cases and therefore be efficient as a preconditioner. We note that given the assumptions made here about the matrix P, D^{-1} is diagonal and therefore we will refer to it in the following either as the standard or diagonal preconditioner. In the reminder of this paper we will show that while the performance of this preconditioner is often very good, better, more general preconditioners can be constructed potentially leading to significant performance gains.

III. Two-level preconditioner for the map-making problem

To motivate the choice of a two-level preconditioner for this research, let us first have a brief look at strengths and weakness of the standard diagonal preconditioner D^{-1}. Some insight into this can be obtained by analyzing the spectrum of the preconditioned system coefficient matrix, i.e., $D^{-1}S$, and compare it with the spectrum of the matrix S alone. An example of such a comparison computed for one of the experiments studied in section V is shown in figure 1. As it is clear from the figure, the preconditioner performs very well as far as the very large eigenvalues of the initial matrix are concerned, suppressing them uniformly to nearly a unity. At the same time, the small eigenvalues of the preconditioned matrix become smaller, and continue to hinder the convergence of the iterative method.

Several solutions exist in the literature to deal with this kind of problem, those include deflation techniques as for instance described in [7] and two-level preconditioners (for a detailed description see for example [8]). These projection methods are based on the usage of a coarse space that incorporates approximations of the eigenvectors corresponding to the small eigenvalues. Let r be the size of the coarse space (usually $r << p$), and let $Z \in \mathbb{R}^{p\times r}$ be the tall and skinny matrix formed by these r vectors. Consider the following matrices, as defined for instance in [8], where S is symmetric and positive definite (SPD) and E is an invertible matrix,

$$\begin{aligned} R &= I - SQ, \\ Q &= ZE^{-1}Z^T, \\ E &= Z^T S Z\,. \end{aligned}$$

The matrices Z, Q, E, and R have different meanings depending on the methods in which they are used. For example in domain decomposition methods, Z and Z^T are restriction and prolongation operators respectively, E is the coarse grid matrix, and Q provides a coarse grid correction. In deflation methods, P is the deflation matrix, and Z contains the vectors to be deflated. These matrices satisfy several equalities, and they allow the definition of several two level preconditioners, for more details see [8], [9]. Our two level preconditioner is based on the adapted deflation preconditioner introduced in [8] (variant 1, referred to as P_{A-DEF1} in that paper),

$$\mathcal{M}_{2lvl} \;=\; D^{-1}R + ZE^{-1}Z^T\,, \quad (12)$$

and combines the effect of the diagonal preconditioner (11) on the higher part of the spectrum with the coarse space correction of the lower part of the spectrum. Compared to the diagonal preconditioner D^{-1}, the two-level preconditioner introduces unavoidably an additional numerical cost, which is however manageable as it corresponds to solving a small linear system with the matrix E (see section IV for more details).

There are different options used in the literature for constructing the matrix Z. In deflation methods, it is formed typically by approximations of eigenvectors that correspond to small eigenvalues of the preconditioned operator $D^{-1}\,S$. The

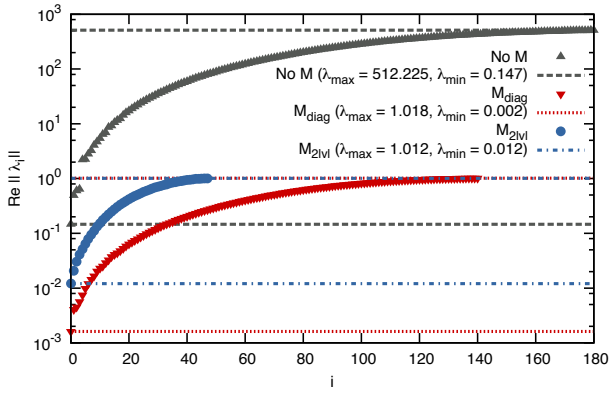


Fig. 1: An example of the approximated spectrum of the matrix S preconditioned with different preconditioners. In the plot, *No M* denotes the case of the matrix S with no preconditioner, M_{diag} marks the case of the matrix S preconditioned by the diagonal matrix D^{-1}, while M_{2lvl} is the matrix preconditioned with our two-level preconditioner.

approximations are based either on some *a priori* knowledge of the problem to be solved, or they are computed *a posteriori*, that is after several iterations of the linear solver with the matrix $D^{-1}\,S$ have been performed. Two-level preconditioners originating from domain decomposition or multigrid methods use for the coarse space a set of projection vectors, which represent an interpolation between fine- and coarse-grid subspaces, and which are built from the coefficients of the input matrix (see for instance [10], [11] and references therein). Here, inspired by these ideas, we propose an *a priori* construction of the coarse space suitable for the generalized least squares problem arising from cosmological observations.

When the instrumental noise $\breve{n}$ is piece-wise stationary as in our application, its noise covariance matrix N^{-1} defined in equation (4) is block-diagonal with K blocks, one per each stationary time interval. Let us then partition the time domain data vector d correspondingly into K sub-vectors $d^1, \ldots, d^K$, that is d^j corresponds to the j-th stationary time interval. The coarse-grid is built by dedicating one column of Z to each piece-wise stationary time interval. Each such column has one nonzero element for each pixel observed during a given time interval. Elements of Z corresponding to a given pixel p_i are then defined to reflect the fact that it has been observed within different time intervals and read as,

$$z(i,j) = \frac{s_i^j}{s_i}, \tag{13}$$

where s_i^j is the number of observations of pixel p_i during the j-th time interval, and s_i is the total number of observations of pixel p_i. In other words, the value $z(i,j)$ represents the fractional number of observations of the pixel p_i in the j-th stationary time interval, which in turn correspond to the sub-vector d^j. Consequently every row of Z represents a partition of the unity, i.e.,

$$Z \;=\; \begin{bmatrix} \frac{s_0^0}{s_0} & \frac{s_0^1}{s_0} & \cdots & \frac{s_0^K}{s_0} \\ \frac{s_1^0}{s_1} & \frac{s_1^1}{s_1} & \cdots & \frac{s_1^K}{s_1} \\ \vdots & \vdots & & \vdots \\ \frac{s_p^0}{s_p} & \frac{s_p^1}{s_p} & \cdots & \frac{s_p^K}{s_p} \end{bmatrix}. \tag{14}$$

With our construction, the nonzero values of Z reflect the dependance between the observations from different piece-wise stationary time intervals. It uses the concept of dependence and influence for building a coarse space, common in algebraic multigrid methods (AMG) [12] and it has similarities with the approach proposed in [13]. As displayed in figure 1, the two-level preconditioner is efficient in improving the condition number of the matrix, by combining the effect of the diagonal preconditioner D^{-1} on the higher part of the spectrum with the coarse space correction of the lower part of the spectrum. In addition, by constructions (equations (13) and (14)), the elements of any single row of Z sum up to 1, and hence the vector $[1, \ldots, 1]$ belongs to the space spanned by the columns of Z. This property could be exploited to construct an alternative family of two-level preconditioners, for which better theoretical understanding of their convergence should be possible. We leave this for future work.

IV. Parallel design of the two-level preconditioner

To benefit from our two-level preconditioner in parallel we need an efficient algorithm for applying the operator $\mathcal{M}_{2lvl}$ defined in equation (12) and the operator S defined in equation (8) to a vector distributed over several processors. Both of these are necessary to compute the Krylov subspace of the form

$$\mathcal{K}_m(A, r_0) = \text{span}\left\{r_0, \mathcal{M}_{2lvl} S r_0, \ldots, (\mathcal{M}_{2lvl} S)^{m-1} r_0\right\}, \tag{15}$$

while iteratively solving the system. (Here $r_0 = b - S\widetilde{x}_0$ is the starting vector of the iterative process.) In this section we describe the parallel design of a numerical routine accomplishing such a task. We also show that thanks to its parallel implementation, the construction of the two level preconditioner can be performed efficiently. If the number of the stationary time intervals is equal to the number of processors, the construction of the coarse space E requires only $\max(n_{row})$ matrix-vector products with the operator S, followed by one call to the `MPI_AllReduce()` routine at the end, where $\max(n_{row})$ is the maximum number of nonzero entries per row of Z. This is typically very small, and at most equal to r, the number of vectors in our coarse space.

Without loss of generality, we take as an example three processors (executed by using three MPI processes). The data vector d is distributed among these three processors and d^j denotes a subvector assigned to a processor j ($1 \leq j \leq 3$).

Following this vector distribution, the matrices N^{-1} and P are stored blockwise. That is, the diagonal block T_j of the matrix N^{-1} is assigned to processor j as well as the corresponding j-th row block of P (referred to as P_j). This distribution allows an efficient implementation of the application of the operator $S = P^T N^{-1} P$ to a current vector of the pixel map x. The matrix-vector products involved here are performed efficiently in parallel, and a more detailed description of the application of the operator S to a vector can be found in [1]. It requires communication only when applying P_j^T to combine values of partial results in the pixel domain from all the processors. This operation can be performed by calling `MPI_AllReduce()` [1], and this is the approach implemented here. We note, however, that recently more efficient solutions have been proposed [6], which scale better with growing number of processors, and could further improve our runtime results.

In the rest of the section we focus on the construction and the application of our two level preconditioner. Following our three block example, we define additionally a five pixel map x, and a pointing matrix P of a corresponding size,

$$x = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix}, \; P^T = \left[\begin{array}{cc|ccc|cc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]. \quad (16)$$

The vertical lines in the matrix P^T denote the distribution of the stationary time intervals between the three processors, which also define the partitioning of the vector d and the block diagonal matrix N^{-1}. The sparse rectangular matrix Z can be build from the values of the matrix P by using the definition (13), and we obtain

$$Z = \begin{bmatrix} z(1,0) & 0 & 0 \\ z(2,0) & z(2,1) & 0 \\ 0 & z(3,1) & 0 \\ 0 & z(4,1) & z(4,2) \\ 0 & 0 & z(5,2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}. \quad (17)$$

Note that due to the sparsity of the matrix P, each processor requires only a part of the map x, that is processor j requires only those pixels whose corresponding columns in P_j are nonzero. Following our example, we introduce now a distributed map $\widetilde{x}$ and a distributed coarse space $\widetilde{Z}$ with which each processor stores only the relevant information from x and Z,

$$\widetilde{x} = \left[\begin{array}{c} p_1 \\ p_2 \\ \hline p_2 \\ p_3 \\ p_4 \\ \hline p_4 \\ p_5 \end{array}\right], \; \widetilde{Z} = \left[\begin{array}{ccc} z(1,0) & 0 & 0 \\ z(2,0) & z(2,1) & 0 \\ \hline z(2,0) & z(2,1) & 0 \\ 0 & z(3,1) & 0 \\ 0 & z(4,1) & z(4,2) \\ \hline 0 & z(4,1) & z(4,2) \\ 0 & 0 & z(5,2) \end{array}\right]. \quad (18)$$

A. Construction of the coarse space operator

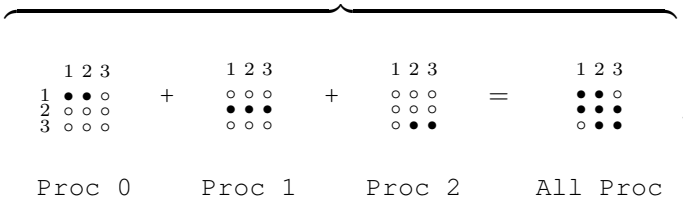
To construct the two level preconditioner, we need to compute the matrix

$$E = Z^T S Z = (Z^T P^T N^{-1} P Z).$$

By using the distributed matrices N^{-1}, P, and Z, this operator can be built from the locally evaluated operators $\widetilde{E}_j$, that is

$$E = \sum_j \widetilde{E}_j = \sum_j (\widetilde{Z}_j^T P_j^T T_j P_j \widetilde{Z}_j), \quad (19)$$

where subscript j indicates the parts of the operators stored by processor j. The sum in equation (19) can be computed by using a collective `MPI_Allreduce()` operation, which can be visualised for our example as follows,



The $\bullet$ and $\circ$ correspond to nonzero and zero values respectively. We will see that the cost of evaluating $\widetilde{E}_j$ is equivalent to computing a number of matrix-vector products equal to the number of nonzero columns in $\widetilde{Z}_j$. Continuing the example from equation (18), we have,

$$\widetilde{W} = \left[\begin{array}{cc} P_0^T T_0 P_0 \widetilde{Z}_0(:,0) & P_0^T T_0 P_0 \widetilde{Z}_0(:,1) \\ \hline P_1^T T_1 P_1 \widetilde{Z}_1(:,0) & P_1^T T_1 P_1 \widetilde{Z}_1(:,1) \quad P_1^T T_1 P_1 \widetilde{Z}_1(:,2) \\ \hline & P_2^T T_2 P_2 \widetilde{Z}_2(:,1) \quad P_2^T T_2 P_2 \widetilde{Z}_2(:,2) \end{array}\right], \quad (20)$$

where the operator $\widetilde{W} = S\,\widetilde{Z}$ is the approximated matrix (SZ), computed in such a way that expensive calls of `MPI_Allreduce()` are omitted whenever P^T is applied on the distributed column of $\widetilde{Z}$ (as part of applying S on a vector). Therefore, rows of $\widetilde{W}$ are build in parallel, only from local operations $P_j^T T_j P_j \widetilde{Z}_j$. And finally

$$\widetilde{E} = \left[\begin{array}{cc} \widetilde{Z}_0^T(:,0)\,\widetilde{W}_0(:,0) & \widetilde{Z}_0^T(:,1)\,\widetilde{W}_0(:,1) \\ \hline \widetilde{Z}_1^T(:,0)\,\widetilde{W}_1(:,0) & \widetilde{Z}_1^T(:,1)\,\widetilde{W}_1(:,1) \quad \widetilde{Z}_1^T(:,2)\,\widetilde{W}_1(:,2) \\ \hline & \widetilde{Z}_2^T(:,1)\,\widetilde{W}_2(:,1) \quad \widetilde{Z}_2^T(:,2)\,\widetilde{W}_2(:,2) \end{array}\right],$$

where $\widetilde{Z}_j(:,k)$ ($\widetilde{W}_j(:,k)$) is a k-th column of $\widetilde{Z}_j$ ($\widetilde{W}_j$).

B. Application of $\mathcal{M}_{2lvl}$ to a vector

Based on the definitions from the previous subsection, we can now elaborate on how to apply $\mathcal{M}_{2lvl}$ to a vector in the pixel domain. This is done by performing the following computation,

$$v_{out} = \left[D^{-1}(I - SZE^{-1}Z^T) + ZE^{-1}Z^T\right] v_{in},$$

where v_{in} is the vector resulting from the application of the operator S to the vector of the current solution of the pixel map. In practice, however, we would like to apply this operator to the distributed map vectors $\widetilde{x}$ with duplicated pixel values, defined as in equation (18). Hence, the above equation is reformulated as follows,

$$\widetilde{v}_{out} = \left[D^{-1}(I - \widetilde{W}E^{-1}\widetilde{Z}^T) + \widetilde{Z}E^{-1}\widetilde{Z}^T\right] \widetilde{v}_{in}. \quad (21)$$

The application of the preconditioner $\mathcal{M}_{2lvl}$ to the vector $\widetilde{v}_{in}$ requires the following computations (corresponding to reading equation (21) from right to left),

- compute the matrix-vector multiplication $\widetilde{Z}^T\widetilde{v}_{in} = v_{small} \in \mathbb{R}^r$,
- compute the operation $E^{-1}v_{small}$, which requires solving the small linear system $Ev^{out}_{small} = v_{small}$,
- and apply the diagonal preconditioner D^{-1} to the distributed map vector.

C. Cost of construction and application of the preconditioner

In the operations listed above, only the computation of $\widetilde{Z}^T\widetilde{v}_{in}$ requires communication in order to sum-reduce the contributions to $v_{small} \in \mathbb{R}^r$ from all the processors. Due to the small size of the output vector, this communication is less expensive than the collective communication required for the application of the operator P^T to a vector.

To reduce the cost of solving the linear system with the matrix E, we first factor it using the LU decomposition. Note that in our algorithm, each processor stores only one copy of the matrix E, even when there are more than one diagonal blocks per processor. Moreover, each processor consists of several cores (six in our experiments) so that we can use multithreaded direct solvers for this task. In summary, the cost of constructing $\mathcal{M}_{2lvl}$ is estimated to be the sum of the cost required for building and decomposing the operator E and the cost of precomputing $\widetilde{W}$ from W. In addition, in practical applications as the CMB map-making process, the same linear system may need to be solved many times with different right hand sides and hence the same preconditioner could be used multiple times.

V. Numerical Experiments

To investigate the performance of our two-level preconditioner we use simulated data with characteristics inspired by existing CMB experiments. We have assumed a scan pattern made of 2048 densely crossing circles as shown in figure 2a and which are scanned one-by-one, consecutively, from left to right. The circles have opening angle of 30°. The time spent on each circle is assumed to be the same. Each circle is scanned 32 times and sampled $\approx 10^6$ times before the experiment switches to another one. As a result of such a procedure we have a vector of more than two billions of observation directions. We note that such a scanning strategy though idealized could be adequate for a satellite-borne observatory.

The sky contribution to the synthetic data vector, d, is then produced for such a scan using a simulated, pixelized CMB map by assigning to each sample a value read off from the map in a sky pixel corresponding to a relevant observation direction. To simulate instrumental noise we assume that the noise is piece-wise stationary with the stationary pieces corresponding to the time intervals spent on each circle. The noise correlations are then present only between the samples corresponding to the same circle and are assumed to be given by Toeplitz matrices with a characteristic correlation length given by so called f_{knee} parameter equal to either 0.5 or 1.5Hz for odd and even circles respectively, corresponding to weaker and stronger noise correlations as shown in figure 2b. The resulting inverse noise covariance, equation (4), is then composed of as many blocks as circles. We use it to generate a random realization of the Gaussian, correlated noise, $\check{n}$ and add it to the sky signal simulated earlier.

To study the impact of using our two-level preconditioner with respect to the diagonal preconditioner, we also consider linear systems corresponding to subsets of the full simulated data set.

A. Comparison with D^{-1}

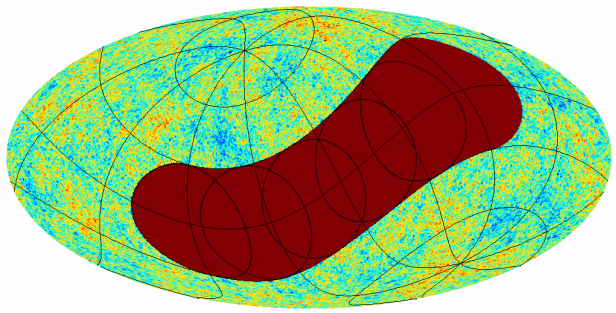
Figure 3 displays the convergence of the preconditioned CG solver for several different data sets, when the number of the circles (and hence the number of the diagonal blocks in N^{-1}) is increased from 8 to 2048. The plot shows the number of steps needed to achieve convergence to a relative error of $(||b - S\widetilde{x}_i||_2/||b||_2) \leq 10^{-7}$ for our two-level preconditioner (blue lines) and for the standard diagonal preconditioner (red lines) (equation (11)). We can see that the convergence rate of the diagonal preconditioner quickly deteriorates, and more iterations are needed, when the number of the circles, and thus the size of the problem, is increased. In contrast, our two-level preconditioner achieves the convergence in about 50 steps for all the different data sets. This shows that our preconditioner is numerically scalable and exhibits robustness with respect to the growing size of the problem.

B. Performance analysis

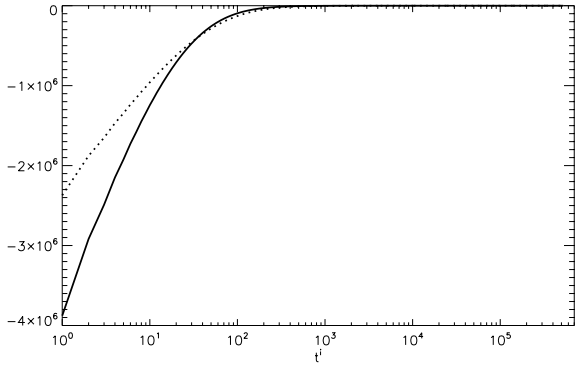
We evaluate the performance of our solver on NERSC's supercomputer Hopper [1], based on a Cray XE6 system. Hopper is formed by 6384 compute nodes, each node is composed of two twelve-cores AMD MagnyCours (2.1 GHz), with a peak performance of 201.6 Gflops per node and 1.288 Peta-flops for the entire machine. Cray's Gemini is the high performance interconnect used for the network.

We use for our tests our own parallel code written in C++/MPI. It is a self contained numerical library, but for the

[1] http://www.nersc.gov/nusers/systems/hopper2/



(a) Simulated reference CMB sky signal with an observed area, corresponding to the full length scan as considered here and made of 2048 crossing circles, highlighted. Five of the scan circles are shown as examples by solid lines.



(b) Profiles of Toeplitz rows. The solid line is for $f_{knee} = 1.5$, while the dotted for $f_{knee} = 0.5$. Only off-diagonal elements are shown, which define the strength and the range of the non-local noise correlations.

Fig. 2: Input data for simulation

sake of better performance third party libraries are used for Fast Fourier transforms required for the Toeplitz algebra [5], and for solving small linear systems with the coarse operator E (see section IV-C). Consequently, we employ multithreaded version of `FFTW` library [14] for FFTs, and Intel's Math Kernel Library for dense linear algebra. Intel Compiler version 12.1.4 is used for compilation.

In all our experiments, all data corresponding to any single circle (in turn corresponding to a single diagonal block of the matrix N^{-1}) are processed by one MPI process, following the distribution described in section IV. Consequently, in our weak scaling performance tests of the preconditioner we map one circle onto one MPI process and thus tying up the problem size and the number of employed MPI processes, which always increase in unison. In order to test strong scaling we fix the number of circles to 1024 and we distribute the data evenly among the available processes by assigning multiple blocks to one MPI process.

Some routines in our package and the `FFTW` library used in our implementation allow us to exploit multithreading when one MPI process is assigned to a multicore processor. Such hybrid codes typically use one OpenMP thread per compute core. Therefore the maximum number of threads per node on Hopper is 24. OpenMP performance can be very dependent on the underlying architecture, but on Hopper, NERSC has found an optimal assignment which we applied in all our runs. It suggests the use of 4 MPI processes per node and 6 OpenMP threads per MPI process. Therefore, we use a maximum of $2048 \times 6 = 12,288$ cores.

For each run we measure the time required for constructing the preconditioner and the time spent in the iterative solver. Additionally, we also keep track of time spent in applying the operators P_j, P_j^{-1}, T_j, and E^{-1} to corresponding vectors during the iterative process, obtaining breakdown of the time spent during a single iteration between the different algebraic operations.

Figure 4 displays the time needed to construct (dashed lines) the two preconditioners (red for D^{-1} and blue for M_{2lvl}) and the time required by the iterative solver to converge to a given tolerance (solid lines). The number of MPI processes increase from 8 to 2048 for the weak scaling experiment and from 32 to 1024 for the strong scaling experiment.

It can be seen from the weak scaling results presented in figure 4a that the time of solving the map-making problem with our two-level preconditioner grows only very slowly with a number of the MPI processes and it is always shorter than the quickly growing time required for solving the same problem with the standard preconditioner. Figure 5a shows the speedup obtained in the iterative process when the two-level preconditioner is used with respect to the usage of the standard preconditioner alone, which can be as large as a factor of 6. When the time of construction of the preconditioner is included, (dashed lines in figure 5a), our method is up to 5 times faster than the standard approach for the largest data sets. Moreover, note that successive linear systems with the same matrix but different right hand sides need to be solved typically in our application, and hence the cost of building the preconditioner may need to be incurred only once.

Strong scaling results are presented in figure 4b, where the problem size is fixed and the number of MPI processes changes from 32 to 1024. The time of solving decreases linearly with a growing number of processes for both preconditioners. Moreover, since each process has one copy of the coarse operator E, the time spent in computing its LU factorization stays constant, hence the speedup obtained when the time of construction of the preconditioner is included is more important when the number of processes is small. In this case the solve time is much more important than the factorization of the coarse operator (see figure 5b). Furthermore, in this experiment, whenever it is possible (e.g., unpointing and pointing) each MPI process performs operations on blocks in parallel using threads. When the time of constructing the preconditioner is not included, our two level preconditioned solver is faster by a factor between 5.7 and 5.5 than the tradi-

tional solver preconditioned by a diagonal matrix. Even with the preconditioner construction time included, the speedup obtained with respect to the solver preconditioned by the diagonal matrix is still very important, between 5.2 and 4.8.

The dashed lines in figure 6 display the overall time breakdown between the main steps of one iteration of the preconditioned system. More than 80% of the overall time is spent in the multiplication of the Toeplitz matrices by a vector, performed by using a parallel (multithreaded) FFT algorithm (green line). The second most expensive operation is the un-pointing operation, $P_j^T \widetilde{x}_j$, which is the only operation which requires communication, performed by using `MPI_Allreduce`. Finally solving the small factorized linear system has almost negligible impact on the total time, with the exception of the two largest test cases, where the multithreaded dense solve becomes almost as expensive as the un-pointing operation. The important observation is that all computations, in the range of sizes tested in our experiment, scale well with a growing number of processors. However, we expect that on a larger number of processors, `MPI_Allreduce` will become less efficient and the communication can become an important fraction of the overall runtime. Better algorithms without such shortcomings have been already proposed, e.g., [6], and will be used in the future. As it can be seen in figure 6 , the time per iteration of the CG solver preconditioned by our two level preconditioner is almost the same as the time per iteration of the CG preconditioned by the standard preconditioner as the solid red and dashed blue lines nearly overlap. This explains the good performance of our preconditioner, which at the same time requires a smaller number of iterations, and hence a smaller overall time to solution.

Finally, to estimate the number of Gflops per iteration, we calculate the number of flops per iteration by using the following equations,

$$F_{M_{diag}}(K, t_j) = K \left[\overbrace{2t_j \, log_2(2t_j)}^{FFT} + 2t_j \right], \quad (22)$$

$$F_{M_{2lvl}}(K, t_j) = F_{M_{diag}}(K, t_j) + K \underbrace{2K^2}_{LUSolve}, \quad (23)$$

where K is the number of the scan circles used in the simulation and t_j is an average number of time samples per ring. The overall number of Gflops is displayed in figure 7, and it is essentially driven by the performance of the (third-party) FFT routine.

In summary, our current implementation of the map-making library allows us to demonstrate the efficiency of our preconditioner. We do not compare it in this paper with MADmap, the state of the art software for solving the map-making problem [1], which uses the diagonal preconditioner, but has several other optimizations such as faster collective communications. However, while we plan in our future work to introduce these optimizations in our library, we estimate that they will equally affect the performance of both preconditioners. Therefore, as long as the cost of solving the system with the factorized matrix E will remain subdominant with respect to the remaining steps of algorithm, the improved routines should preserve the speedups measured in our experiments. Indeed, for most of the real experiments, the coarse operator E tends to be a sparse matrix (see section IV). Therefore, it may be beneficial to use a direct method for sparse matrices as SuperLU [15] or MUMPS [16].

VI. Conclusions

This paper describes a parallel two-level preconditioner for the cosmological map-making problem. Our iterative algorithm is based on a conjugate gradient method with a parallel two-level preconditioner (2lvl-PCG) for which we construct its coarse space from a limited number of sparse vectors estimated solely from coefficients of the initial linear system. We show experimentally that in the context of simulated Cosmic Microwave Background data sets, our parallel implementation of 2lvl-PCG outperforms the standard one-level PCG in terms of both the convergence rate and time to the solution by as much as a factor of 6 on up to $12,288$ cores and maintaining nearly perfect scalability, weak and strong, in such a regime.

Acknowledgments

This work has been supported in part by French National Research Agency (ANR) through COSINUS program (project MIDAS no. ANR-09-COSI-009). The HPC resources were provided by GENCI- [CCRT/TGCC] (grants 2011-066647 and 2012-066647) in France and by the NERSC in the US, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] C. M. Cantalupo, J. D. Borrill, A. H. Jaffe, T. S. Kisner, and R. Stompor, "MADmap: A Massively Parallel Maximum Likelihood Cosmic Microwave Background Map-maker," *The Astrophys. Journal Suppl. Series*, vol. 187, pp. 212–227, Mar. 2010.

[2] E. L. Wright, G. Hinshaw, and C. L. Bennett, "Producing megapixel cosmic microwave background maps from differential radiometer data," *The Astrophysical Journal Letters*, vol. 458, no. 2, p. L53, 1996.

[3] M. Tegmark, "How to make maps from cosmic microwave background data without losing information," *The Astrophysical Journal Letters*, vol. 480, no. 2, p. L87, 1997.

[4] R. Stompor, A. Balbi, J. D. Borrill, P. G. Ferreira, S. Hanany, A. H. Jaffe, A. T. Lee, S. Oh, B. Rabii, P. L. Richards, G. F. Smoot, C. D. Winant, and J.-H. P. Wu, "Making maps of the cosmic microwave background: The MAXIMA example," *Phys. Rev. D*, vol. 65, no. 2, p. 022003, Jan. 2002.

[5] G. Golub and C. Van Loan, *Matrix computations*. Johns Hopkins Univ Pr, 1996, vol. 3.

[6] R. Sudarsan, J. Borrill, C. Cantalupo, T. Kisner, K. Madduri, L. Oliker, Y. Zheng, and H. Simon, "Cosmic microwave background map-making at the petascale and beyond," in *Proceedings of the international conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 305–316.

[7] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h, "A deflated version of the conjugate gradient algorithm," *SIAM J. Sci. Comput.*, vol. 21, no. 5, pp. 1909–1926 (electronic), 2000, iterative methods for solving systems of algebraic equations (Copper Mountain, CO, 1998).

[8] J. M. Tang, R. Nabben, C. Vuik, and Y. A. Erlangga, "Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods," *J. Sci. Comput.*, vol. 39, pp. 340–370, 2009.

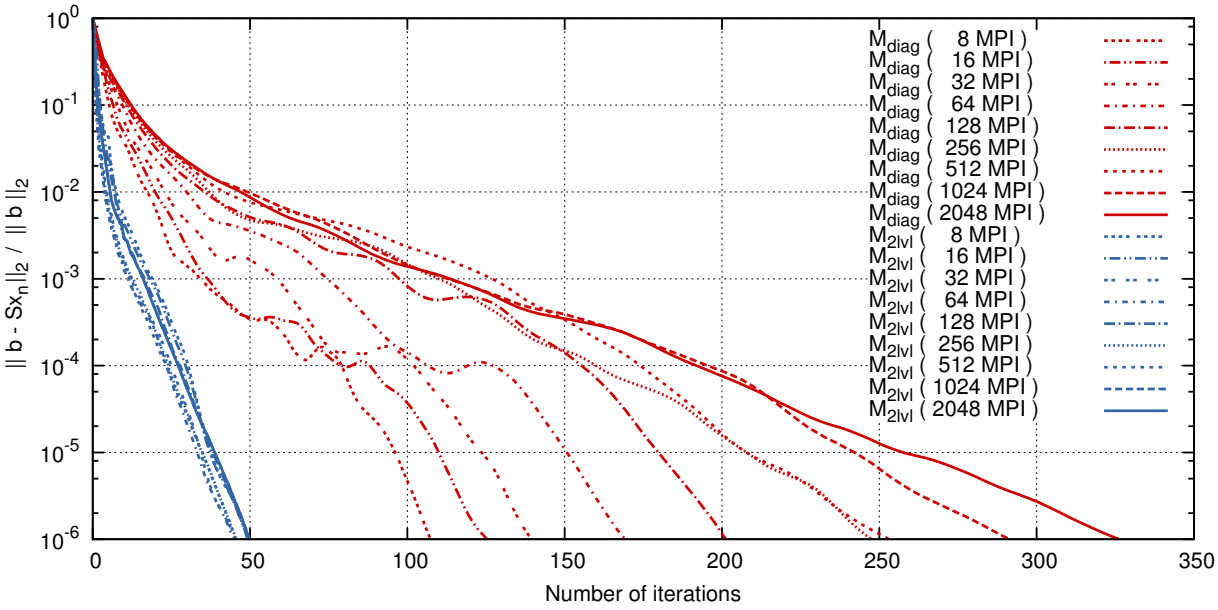
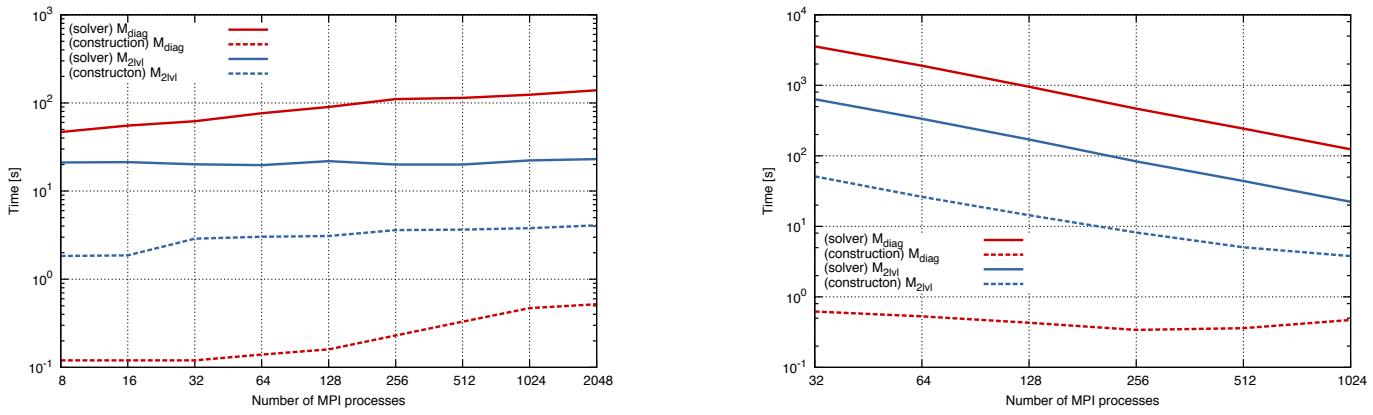

|| b - Sx_n||_2 / || b ||_2
Number of iterations
M_diag (8 MPI)
M_diag (16 MPI)
M_diag (32 MPI)
M_diag (64 MPI)
M_diag (128 MPI)
M_diag (256 MPI)
M_diag (512 MPI)
M_diag (1024 MPI)
M_diag (2048 MPI)
M_2lvl (8 MPI)
M_2lvl (16 MPI)
M_2lvl (32 MPI)
M_2lvl (64 MPI)
M_2lvl (128 MPI)
M_2lvl (256 MPI)
M_2lvl (512 MPI)
M_2lvl (1024 MPI)
M_2lvl (2048 MPI)

Fig. 3: Convergence of the preconditioned CG solver of the CMB map-making problem applied in the case of the simulated data as described in section V. Different types of lines correspond to different sizes of the problem, while different colors correspond to the two different types of preconditioners: red for the standard preconditioner D^{-1}, and blue for the two-level preconditioner M_{2lvl} proposed in this work. In the legend, the numbers in parenthesis corresponds to the number of MPI processes used in the experiments and are equal to the number of diagonal blocks in N^{-1}.



(a) Weak scaling

(b) Strong scaling

Fig. 4: Overall time spent in the construction of the preconditioners and in the iterative solver for the diagonal preconditioner M_{diag} and the new two-level preconditioner M_{2lvl}.

[9] J. Tang, R. Nabben, C. Vuik, and Y. Erlangga, "Theoretical and numerical comparison of various projection methods derived from deflation, domain decomposition and multigrid methods," Delft University of Technology, Delft Institute of Applied Mathematics, Delft, Report 07-04, 2007.

[10] D. M. Alber and L. N. Olson, "Parallel coarse-grid selection," *Numerical Linear Algebra with Applications*, vol. 14, no. 8, pp. 611–643, 2007.

[11] V. E. Henson and U. M. Yan, "BoomerAMG: A parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155 – 177, 2002.

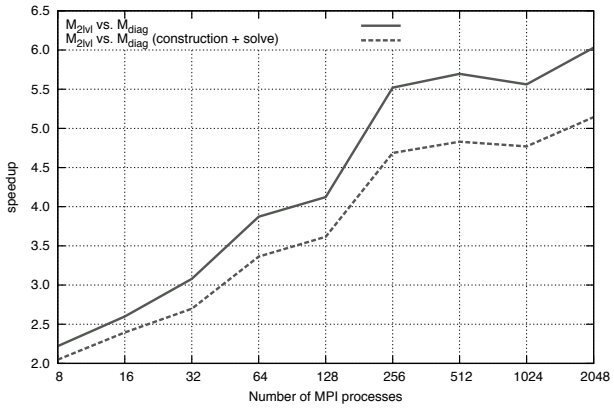
[12] V. E. Henson and U. M. Yang, "BoomerAMG: a parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, pp. 155–177, 2000.

[13] R. A. Nicolaides, "Deflation of conjugate gradients with applications to boundary value problems," *SIAM Journal on Numerical Analysis*, vol. 24, no. 2, pp. 355–365, Apr. 1987.

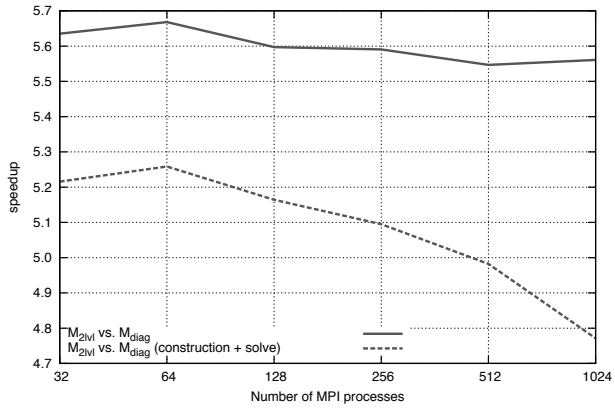
[14] M. Frigo and S. G. Johnson, "FFTW: an adaptive software architecture for the FFT," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 3. IEEE, May 1998, pp. 1381–1384 vol.3.

[15] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, "A supernodal approach to sparse partial pivoting," *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 3, pp. 720–755, 1999.

[16] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

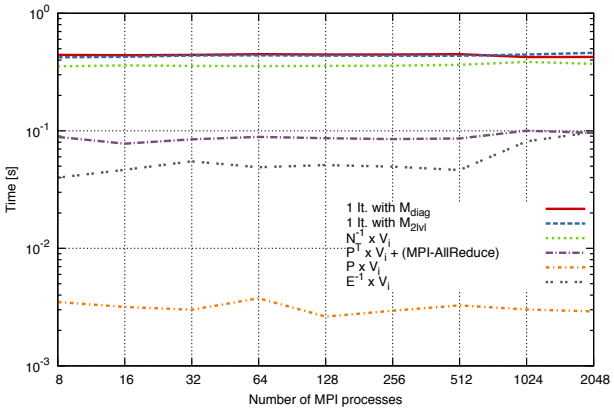


(a) Weak scaling

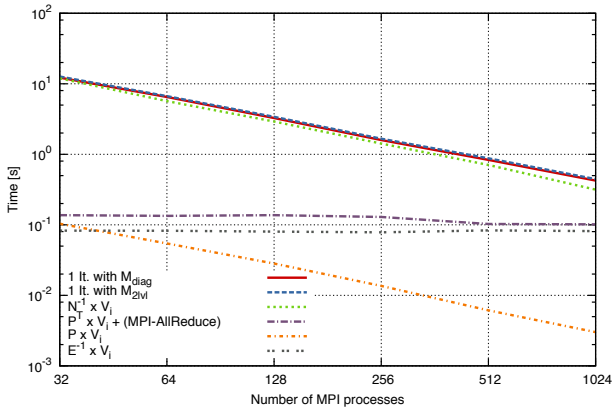


(b) Strong scaling

Fig. 5: Speedup of the two level method preconditioner with respect to the traditional solver with the diagonal preconditioner.

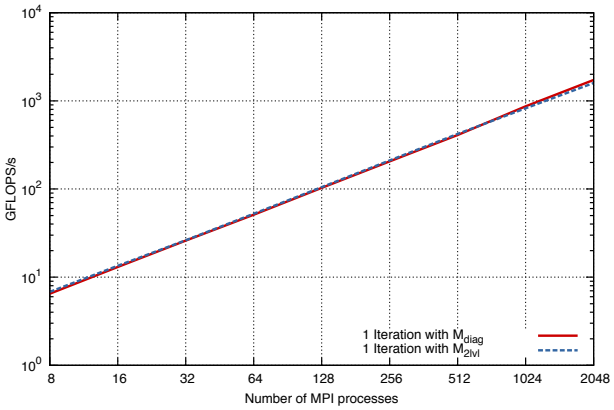


(a) Weak scaling

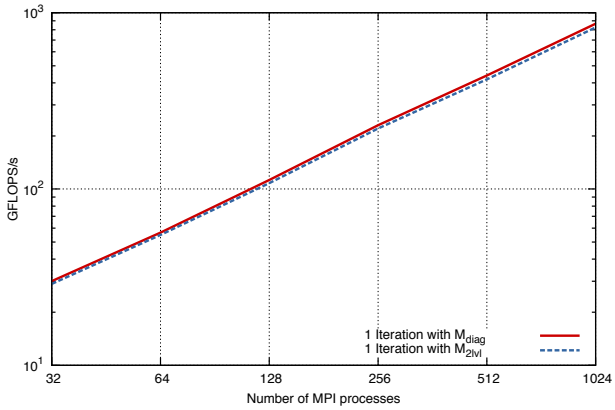


(b) Strong scaling

Fig. 6: Average time per iteration and a time breakdown into the main operations.



(a) Weak scaling



(b) Strong scaling

Fig. 7: Average number of GFlops per iteration as a function of the size of the problem. This number is mostly driven by the FFT libraries used to perform the product of the inverse noise covariance matrix and a vector.